CLAIMS

What is claimed is:

1. A method of debugging a remote computer, comprising:

   running a debugger on a host computer;

   running an operating system on the target computer;

   when debugging is required, loading a debug agent from persistent store into memory and executing the debug agent;

   executing initialization code of said debug agent, wherein replacing selected OS kernel code and data that are referenced, accessed, and otherwise used in the processing of debugging traps by the OS kernel, and whereas said replaced code and data reside in or reference to said debug agent code and data images in memory;

   while the target is being debugged, the debug agent intercepting and processing one or more processor debugging traps generated;

   when debugging is no longer required, unloading the debug agent, wherein restoring replaced OS kernel and data to original values.

2. The method of claim 1, wherein replaced OS code and data comprising:

   a. private code and data, which are accessible only to references within the OS kernel program image;

   b. exported code and data, which are accessible to references outside and within the OS kernel program image;

   further wherein said references comprising:

   c. one or more application program codes via system calls;

   d. one or more loadable modules or device drivers via exported OS kernel interfaces.

3. The method of claim 1, wherein processing of said debugging traps comprising:

a. invoking the OS kernel private or exported functions by the debug agent;

b. accessing the OS kernel private or exported variables by the debug agent.

4. The method of claim 1, further comprising specifying addresses of OS kernel private code and data to the debug agent by at least one of the steps, comprising:

a. passing messages from host debugger at connect time to the debug agent;

b. passing parameters to the module loading program invoked to load the module.

5. The method of claim 1, wherein the debug agent program image is compiled or linked as code to be dynamically loaded and executed by the OS,

a. wherein the debugged program code executing in a non-exception context,

b. wherein the debugged program code comprising: one or more dynamically loaded or statically linked device drivers; and one or more application programs executing as one or more processes and threads;

c. further wherein non-exception context excludes the processor traps and interrupt handler context.

6. The method of claim 1, wherein said debugging traps are generated by an event from the selected one of:

a. executing a processor BREAK instruction;

b. executing an invalid instruction;

c. executing an instruction causing data access failure;

d. generating any device interrupts or processor traps causing the processor to enter exception context.

7. A method of intercepting the OS loadable module loading system call, comprising:

a. running a debug agent on the target system;

b. saving an entry in the OS syscall table pointing to a sys_init_module function, which services the OS module loading system call;

c. replacing said entry with a pointer to a proxy sys_init_module function residing in the debug agent memory image;

responsive to system call to load the loadable module by the loading utility program, executing the steps comprising:

d. invoking said proxy sys_init_module function via said replacement;

e. responsive to determining that said module has been selected for debugging, initiating debugging of the loadable module by the said proxy sys_init_module function.

8. The method of claim 7, further comprising debugging of an init_module function that is specific to and part of said loadable module, wherein said init_module function is normally invoked by the OS sys_init_module function after the loadable module image is loaded in memory, comprising the steps:

a. saving the pointer to the debugged module init_module function, whereas said pointer is part of the OS kernel data structure for the loadable module;

b. setting said pointer in the OS kernel data structure to a predetermined value denoting the absence of the init_module function for the loadable module;

c. calling the original saved sys_init_module function to load the code and data image of loadable module into memory;

d. setting a breakpoint at entry to the loadable module init_module function using a break code denoting the module inserting event;

e. invoking the loadable module init_module function, triggering said module inserting breakpoint, and invoking the debug agent debugging trap handler to effect debugging of the module.

9. The method of claim 7, further comprising calculating the start address of the loaded module, whenever such information is not available, comprising the steps:

a. notifying the host debugger that the debugged module is loaded, passing the addresses of the module init_module and cleanup_module functions, wherein the OS normally invokes said init_module function after the debugged module is loaded in memory, and further wherein the OS normally invokes said cleanup_module function before the debugged module is unloaded from memory, and whereas a module may have a selected one of one, none, or both init_module and cleanup_module functions specified;

b. responsive to module loading notification, calculating the absolute value of the start address of the in-memory program code area by offsetting the provided addresses of the init_module and cleanup_module functions from their respective relative addresses contained in the symbol table of the debugged module program image file;

c. comparing results of calculation based on each address for validation.

10. The method of claim 9, further comprising:

a. calculating the start address using one such function address available;

b. failing to provide symbolic debugging of the module if none is available.

11. The method of claim 7, further comprising asynchronously putting the target system under debug, comprising the steps:

a. configuring the host debugger to debug a benign module, which does nothing when loaded and unloaded;

b. loading the benign module when the target needs to be debugged;

c. unloading the benign module when resuming the target execution.

12. A method for transferring execution flow from the debug agent exception handler to and from the debug agent command loop after the occurrence of a debugging trap, comprising:

a. prior to the debugging trap occurrence, the debug agent capturing the execution context at the destination within the debug agent command loop;

b. saving of debugged entity context at trap occurrence in the context saved area and invoking the debug agent trap handler;

c. the debug agent trap handler saving and replacing the contents of the context saved area with said captured context at the destination;

d. the debug agent trap handler executing the exception return code to resume system execution to the specified destination within the debug agent command loop, whereas said specified destination context is stored in the context saved area;

e. the debug agent responding to one or more access requests from the host debugger;

the debug agent, responsive to a run-control request, performing steps comprising:

f. setting a global variable to a value denoting transference of command loop to trap handler, whereas such variable is accessible to debug agent command loop and debug agent trap handler;

g. executing an instruction causing the system to enter exception mode, wherein said instruction is a selected one of: a BREAK instruction or an illegal instruction, further wherein the BREAK code or the illegal instruction opcode denotes transference;

h. the system invoking the debug agent trap handler on entrance to exception mode;

responsive to determining that both the value of the global variable and the break code or the illegal instruction opcode denotes transference, the debug agent trap handler resuming execution to the debugged entity, comprising the steps:

i. restoring original execution context of the debugged entity to the context saved area;

j. executing the exception return code, resuming system execution to the destination at or near the point of the debugging trap occurrence in the debugged entity.

13. The method of claim 12, wherein the debug agent exception handler executes under system exception context and the debug agent command loop executes under system non-exception context.

14. The method of claim 12, further comprising communicating with the remote host debugger from the debug agent command loop, comprising the steps:

a. at debug agent initialization time, establishing a socket connection to the host debugger via available networking devices on the target;

b. exchanging debugging messages with the host debugger by invoking OS kernel send and receive functions from the debug agent command loop;

wherein said network devices operate in interrupt-driven mode via the OS kernel built-in device drivers, further wherein network devices consisting: wired and wireless Ethernet, serial, firewire, parallel, and USB devices.

15. The method of claim 12, further comprising communicating with the remote host debugger from the debug agent command loop, comprising the steps:

a. connecting a hardware-assisted debugging probe to the target computer;

b. connecting the host computer to the hardware-assisted debugging probe;

c. exchanging debugging messages by writing and reading dedicated memory regions on the target computer accessible to both the host debugger and the debug agent command loop;

wherein said hardware-assisted debugging probe consisting: JTAG emulator, or ROM emulator.

16. An apparatus comprising:

a target computer, comprising one or more processors;

a memory coupled to the processor;

a hardware bus coupling the processor and one or more peripheral devices;

one or more communicating peripheral devices coupled to the hardware bus;

an operating system running on the processor;

one or more programs, each residing in memory and executing on the processor as one or more processes or threads;

one or more device driver drivers, each loaded by the OS on demand;

a host computer, connecting to the target computer via communicating peripheral devices;

a host debugger executing on the host computer;

a debug agent, loaded by the OS on demand, residing in memory and executing on the target computer, wherein:

a. said debug agent is loaded from persistent store into memory and executed when debugging is required;

b. said debug agent initialization code replacing selected OS kernel code and data that are referenced, accessed, and otherwise used in the processing of debugging traps by the OS kernel, and whereas said replaced code and data reside in or reference to said debug agent code and data images in memory;

c. while the target is under debug, said debug agent intercepting and processing one or more processor debugging traps generated;

d. unloading the debug agent, wherein restoring replaced OS kernel and data to original values when debugging is no longer required.

17. The apparatus of claim 16, wherein replaced OS code and data comprising:

a. private code and data, which are accessible only to references within the OS kernel program image;

b. exported code and data, which are accessible to references outside and within the OS kernel program image;

further wherein said references comprising:

c. one or more application program codes via system calls;

d. one or more loadable modules or device drivers via exported OS kernel interfaces.

18. The apparatus of claim 16, wherein processing of said debugging traps comprising:

a. invoking the OS kernel private or exported functions by the debug agent;

b. accessing the OS kernel private or exported variables by the debug agent.

19. The apparatus of claim 16, wherein the debug agent specifying of OS kernel private code and data to the debug agent by at least one of the steps, comprising:

a. passing messages from host debugger at connect time to the debug agent;

b. passing parameters to the module loading program invoked to load the module.

20. The apparatus of claim 16, wherein the debug agent program image is compiled or linked as code to be dynamically loaded and executed by the OS,

a. wherein the debugged program code executing in a non-exception context,

b. wherein the debugged program code comprising: one or more dynamically loaded or statically linked device drivers; and one or more application programs executing as one or more processes and threads;

c. further wherein non-exception context excludes the processor traps and interrupt handler context.

21. The apparatus of claim 16, wherein said debugging traps are generated by an event from the selected one of:

a. executing a processor BREAK instruction;

b. executing an invalid instruction;

c. executing an instruction causing data access failure;

d. generating any device interrupts or processor traps causing the processor to enter exception context.

22. The apparatus of claim 16, wherein the debug agent intercepting the OS loadable module loading system call, comprising:

a. saving an entry in the OS syscall table pointing to a sys_init_module function, which services the OS module loading system call;

b. replacing said entry with a pointer to a proxy sys_init_module function residing in the debug agent memory image;

responsive to system call to load the loadable module by the loading utility program, executing the steps comprising:

c. invoking said proxy sys_init_module function via said replacement;

d. responsive to determining that said module has been selected for debugging, initiating debugging of the loadable module by the said proxy sys_init_module function.

23. The apparatus of claim 22, further wherein the debug agent debugging of an init_module function that is specific to and part of said loadable module, wherein said init_module function is normally invoked by the OS sys_init_module function after the loadable module image is loaded in memory, comprising the steps:

a. saving the pointer to the debugged module init_module function, whereas said pointer is part of the OS kernel data structure for the loadable module;

b. setting said pointer in the OS kernel data structure to a predetermined value denoting the absence of the init_module function for the loadable module;

c. calling the original saved sys_init_module function to load the code and data image of loadable module into memory;

d. setting a breakpoint at entry to the loadable module init_module function using a break code denoting the module inserting event;

e. invoking the loadable module init_module function, triggering said module inserting breakpoint, and invoking the debug agent debugging trap handler to effect debugging of the module.

24. The apparatus of claim 22, further wherein the debug agent calculating the start address of the loaded module, whenever such information is not available, comprising the steps:

a. notifying the host debugger that the debugged module is loaded, passing the addresses of the module init_module and cleanup_module functions, wherein the

OS normally invokes said init_module function after the debugged module is loaded in memory, and further wherein the OS normally invokes said cleanup_module function before the debugged module is unloaded from memory, and whereas a module may have a selected one of one, none, or both init_module and cleanup_module functions specified;

b. responsive to module loading notification, calculating the absolute value of the start address of the in-memory program code area by offsetting the provided addresses of the init_module and cleanup_module functions from their respective relative addresses contained in the symbol table of the debugged module program image file;

c. comparing results of calculation based on each address for validation.

25. The apparatus of claim 24, further wherein the debug agent:

a. calculating the start address using one such function address available;

b. failing to provide symbolic debugging of the module if none is available.

26. The apparatus of claim 22, further wherein the debug system asynchronously putting the target system under debug, comprising the steps:

a. configuring the host debugger to debug a benign module, which does nothing when loaded and unloaded;

b. loading the benign module when the target needs to be debugged;

c. unloading the benign module when resuming the target execution.

27. The apparatus of claim 16, wherein the debugging system transferring of execution flow from the debug agent exception handler to and from the debug agent command loop after the occurrence of a debugging trap, comprising:

a. prior to the debugging trap occurrence, the debug agent capturing the execution context at the destination within the debug agent command loop;

b. saving of system context at trap occurrence in the context saved area and invoking the debug agent trap handler;

c. the debug agent trap handler saving and replacing the contents of the context saved area with said captured context at the destination;

d. the debug agent trap handler executing the exception return code to resume system execution to the specified destination within the debug agent command loop, whereas said specified destination context is stored in the context saved area;

e. the debug agent responding to one or more access request from the host debugger;

the debug agent, responsive to a run-control request from the host debugger, performing steps comprising:

f. setting a global variable to a value denoting transference of command loop to trap handler, whereas such variable is accessible to debug agent command loop and debug agent trap handler;

g. executing an instruction causing the system to enter exception mode, wherein said instruction is a selected one of: a BREAK instruction or an illegal instruction, further wherein the BREAK code or the illegal instruction opcode denotes transference;

h. the system invoking the debug agent trap handler on entrance to exception mode;

responsive to determining that both the value of the global variable and the break code or the illegal instruction opcode denotes transference, the debug agent trap handler resuming execution to the debugged entity, comprising the steps:

i. restoring original execution context of the debugged entity to the context saved area;

j. executing the exception return code, resuming system execution to the destination at or near the point of the debugging trap occurrence in the debugged entity.

28. The apparatus of claim 27, wherein the debug agent exception handler executes under system exception context and the debug agent command loop executes under system non-exception context.

29. The apparatus of claim 27, wherein communicating with the remote host debugger from the debug agent command loop comprising the steps:

a. at debug agent initialization time, establishing a socket connection to the host debugger via available networking devices on the target;

b. exchanging debugging messages with the host debugger by invoking OS kernel send and receive functions from the debug agent command loop;

wherein said network devices operate in interrupt-driven mode via the OS kernel built-in device drivers, further wherein network devices consisting: wired and wireless Ethernet, serial, firewire, parallel, and USB devices.

30. The apparatus of claim 27, wherein communicating with the remote host debugger from the debug agent command loop comprising the steps:

a. connecting a hardware-assisted debugging probe to the target computer;

b. connecting the host computer to the hardware-assisted debugging probe;

c. exchanging debugging messages by writing and reading dedicated memory regions on the target computer accessible to both the host debugger and the debug agent command loop;

wherein said hardware-assisted debugging probe consisting: JTAG emulator, or ROM emulator.

31. A program product comprising:

a debugger running on a host computer;

a debug agent, when loaded from persistent store into memory and executed; comprising the steps:

a. executing the debug agent initialization, wherein replacing selected OS kernel code and data that are referenced, accessed, and otherwise used in the processing of debugging traps by the OS kernel, and whereas said replaced code and data reside in or reference to said debug agent code and data images in memory;

b. while the target is being debugged, intercepting and processing one or more processor debugging traps generated;

c. when debugging is no longer required, unloading and restoring replaced OS kernel and data to original values.

32. The program product of claim 31, wherein replaced OS code and data comprising:

a. private code and data, which are accessible only to references within the OS kernel program image;

b. exported code and data, which are accessible to references outside and within the OS kernel program image;

further wherein said references comprising:

c. one or more application program codes via system calls;

d. one or more loadable modules or device drivers via exported OS kernel interfaces.

33. The program product of claim 31, wherein processing of said debugging traps comprising:

a. invoking the OS kernel private or exported functions by the debug agent;

b. accessing the OS kernel private or exported variables by the debug agent.

34. The program product of claim 31, further comprising specifying addresses of OS kernel private code and data to the debug agent by at least one of the steps, comprising:

a. passing messages from host debugger at connect time to the debug agent;

b. passing parameters to the module loading program invoked to load the module.

35. The program product of claim 31, wherein the debug agent program image is compiled or linked as code to be dynamically loaded and executed by the OS,

a. wherein the debugged program code executing in a non-exception context,

b. wherein the debugged program code comprising: one or more dynamically loaded or statically linked device drivers; and one or more application programs executing as one or more processes and threads;

c. further wherein non-exception context excludes the processor traps and interrupt handler context.

36. The program product of claim 31, wherein said debugging traps are generated by an event from the selected one of:

a. executing a processor BREAK instruction;

b. executing an invalid instruction;

c. executing an instruction causing data access failure;

d. generating any device interrupts or processor traps causing the processor to enter exception context.

37. A program product for intercepting the OS loadable module loading system call, comprising:

a. running a debug agent on the target system;

b. saving an entry in the OS syscall table pointing to a sys_init_module function, which services the OS module loading system call;

c. replacing said entry with a pointer to a proxy sys_init_module function residing in the debug agent memory image;

responsive to system call to load the loadable module by the loading utility program, executing the steps comprising:

d. invoking said proxy sys_init_module function via said replacement;

e. responsive to determining that said module has been selected for debugging, initiating debugging of the loadable module by the said proxy sys_init_module function.

38. The program product of claim 37, further comprising debugging of an init_module function that is specific to and part of said loadable module, wherein said init_module function is normally invoked by the OS sys_init_module function after the loadable module image is loaded in memory, comprising the steps:

a. saving the pointer to the debugged module init_module function, whereas said pointer is part of the OS kernel data structure for the loadable module;

b. setting said pointer in the OS kernel data structure to a predetermined value denoting the absence of the init_module function for the loadable module;

c. calling the original saved sys_init_module function to load the code and data image of loadable module into memory;

d. setting a breakpoint at entry to the loadable module init_module function using a break code denoting the module inserting event;

e. invoking the loadable module init_module function, triggering said module inserting breakpoint, and invoking the debug agent debugging trap handler to effect debugging of the module.

39. The program product of claim 37, further comprising calculating the start address of the loaded module, whenever such information is not available, comprising the steps:

a. notifying the host debugger that the debugged module is loaded, passing the addresses of the module init_module and cleanup_module functions, wherein the OS normally invokes said init_module function after the debugged module is loaded in memory, and further wherein the OS normally invokes said cleanup_module function before the debugged module is unloaded from memory, and whereas a module may have a selected one of one, none, or both init_module and cleanup_module functions specified;

b. responsive to module loading notification, calculating the absolute value of the start address of the in-memory program code area by offsetting the provided addresses of the init_module and cleanup_module functions from their respective relative addresses contained in the symbol table of the debugged module program image file;

c. comparing results of calculation based on each address for validation.

40. The program product of claim 39, further comprising:

a. calculating the start address using one such function address available;

b. failing to provide symbolic debugging of the module if none is available.

41. The program product of claim 37, further comprising asynchronously putting the target system under debug, comprising the steps:

a. configuring the host debugger to debug a benign module, which does nothing when loaded and unloaded;

b. loading the benign module when the target needs to be debugged;

c. unloading the benign module when resuming the target execution.

42. A program product for transferring execution flow from the debug agent exception handler to and from the debug agent command loop after the occurrence of a debugging trap, comprising:

a. prior to the debugging trap occurrence, the debug agent capturing the execution context at the destination within the debug agent command loop;

b. saving of system context at trap occurrence in the context saved area and invoking the debug agent trap handler;

c. the debug agent trap handler saving and replacing the contents of the context saved area with said captured context at the destination;

d. the debug agent trap handler executing the exception return code to resume system execution to the specified destination within the debug agent command loop, whereas said specified destination context is stored in the context saved area;

e. the debug agent responding to one or more access requests from the host debugger;

the debug agent, responsive to a run-control request, performing steps comprising:

f. setting a global variable to a value denoting transference of command loop to trap handler, whereas such variable is accessible to debug agent command loop and debug agent trap handler;

g. executing an instruction causing the system to enter exception mode, wherein said instruction is a selected one of: a BREAK instruction or an illegal instruction, further wherein the BREAK code or the illegal instruction opcode denotes transference;

h. the system invoking the debug agent trap handler on entrance to exception mode;

responsive to determining that both the value of the global variable and the break code or the illegal instruction opcode denotes transference, the debug agent trap handler resuming execution to the debugged entity, comprising the steps:

i. restoring original execution context of the debugged entity to the context saved area;

j. executing the exception return code, resuming system execution to the destination at or near the point of the debugging trap occurrence in the debugged entity.

43. The program product of claim 42, wherein the debug agent exception handler executes under system exception context and the debug agent command loop executes under system non-exception context.

44. The program product of claim 42, further comprising communicating with the remote host debugger from the debug agent command loop, comprising the steps:

a. at debug agent initialization time, establishing a socket connection to the host debugger via available networking devices on the target;

b. exchanging debugging messages with the host debugger by invoking OS kernel send and receive functions from the debug agent command loop;

wherein said network devices operate in interrupt-driven mode via the OS kernel built-in device drivers, further wherein network devices consisting: wired and wireless Ethernet, serial, firewire, parallel, and USB devices.

45. The program product of claim 42, further comprising communicating with the remote host debugger from the debug agent command loop, comprising the steps:

a. connecting a hardware-assisted debugging probe to the target computer;

b. connecting the host computer to the hardware-assisted debugging probe;

c. exchanging debugging messages by writing and reading dedicated memory regions on the target computer accessible to both the host debugger and the debug agent command loop;

wherein said hardware-assisted debugging probe consisting: JTAG emulator, or ROM emulator.